(19) Europäisch s Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 843 267 A2

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
20.05.1998 Bulletin 1998/21

(51) Int. Cl.[6]: **G06F 17/30**

(21) Application number: **97120021.7**

(22) Date of filing: **14.11.1997**

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **19.11.1996 US 752218**

(71) Applicant: **MICROSOFT CORPORATION**
Redmond, Washington 98052-6399 (US)

(72) Inventors:
- **Bennett, John G.**
  Bellevue, Washington 98004 (US)
- **Dalal, Ketan**
  Seattle, Washington 98102 (US)

(74) Representative:
Grünecker, Kinkeldey,
Stockmair & Schwanhäusser
Anwaltssozietät
Maximilianstrasse 58
80538 München (DE)

(54) **Processing multiple database transactions in the same process to reduce process overhead and redundant retrieval from database servers**

(57) The present invention uses a segmented caching data structure to cache database objects provided by a database server. The database server provides database objects in response to requests by a number of different programs. The segmented caching data structure is made up of a single central cache and a number of program caches, each corresponding to one of the programs. When a database object is provided by the database server in response to a request by any of the programs, a copy of the database object is stored in the central cache. Another copy of the object is stored in the program cache for the program that requested the database object. When the segmented caching data structure is maintained in this manner, when a request is made by one of the programs a copy of the requested object stored in either of the central cache or the program cache for the program may be used, making it unnecessary for the database server to provide the requested database object.

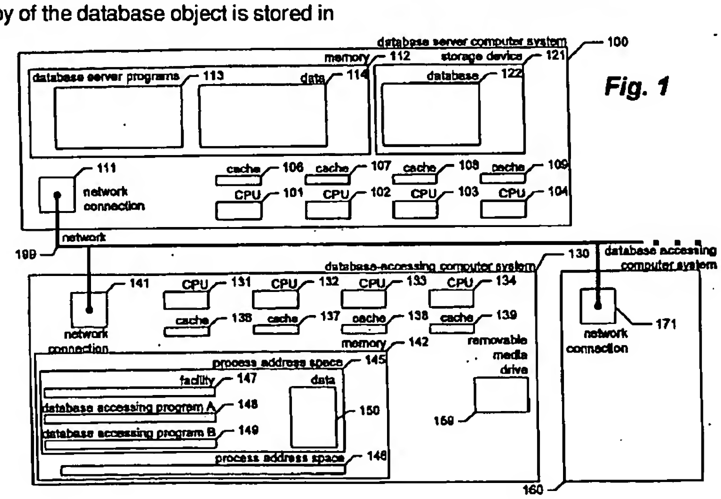


Fig. 1

EP 0 843 267 A2

WEST

Descripti n

Technical Field

The invention relates generally to the fields of database transaction processing and caching the results of retrieval requests.

Background of the Invention

It is common to use databases to store significant quantities of data, especially in cases in which the data is shared by a number of database-accessing programs. Database-accessing programs that access data in a database may be executed on a number of different connected computer system. These programs issue a series of database transactions, each corresponding to one or more operations on the database, including read and write operations.

When two or more such programs are executed on the same computer system, they are typically each executed in a separate process. Each process corresponds to a set of resources provided by the operating system, most notably an addressable range of memory, that is available only to programs ("threads") running within the process. Database-accessing programs are generally each executed in a separate process to prevent them from corrupting each other's data. Because database-accessing programs executing in separate processes cannot share data, the results of a read operation obtained by one database-accessing program are unavailable to other database-accessing programs that issue the same read operations. Indeed, because each database-accessing program typically discards the results of read operation performed as part of a transaction when the transaction completes, a single database-accessing program may have to issue the same read operation two or more times in a short period of time. These redundant transactions again must be applied directly against the database, which has significant time cost. First, the database-accessing program must transmit the transaction across a network to the computer system containing the database, which can take a substantial amount of time. Further, the to actually apply the transaction against the database, the database-accessing program must obtain the appropriate locks, or access controls, on the database, which can involve further network communication and synchronization with database-accessing programs executing on still other computer systems.

Further, because each process has an extensive set of resources devoted to it, the operations of creating and destroying processes each have significant time cost. In the conventional approach of executing each database-accessing program in a separate process, this significant time cost is incurred any time an database-accessing program begins or ends execution. In addition, many of the kinds resources devoted to separate processes are scarce, and allocating shares of these kinds of resources to each of a large number of processes further degrades the performance of the computer system and limits the number of other programs that can simultaneously execute on the computer system.

Given the significant disadvantages of the conventional approach to executing database database-accessing programs, an alternative approach to executing database programs that reduces redundant retrieval from database servers and reduces process overhead would have significant utility.

Summary of the Invention

In accordance with the present invention a database object caching facility ("the facility") maintains a hierarchy of caches that enables database-accessing programs processing database transactions to share retrieved database objects across database transactions, reducing process overhead and redundant retrieval from database servers, while still maintaining read-repeatable transactional isolation. "Database objects" as used herein means any unit of data that may be retrieved from a database including tables, fields, files, programmatic objects, and other units of data.

The facility executes multiple database-accessing programs in the same process, and utilizes a hierarchy of caches to cache database objects retrieved from a database server. The hierarchy of caches includes one program cache for each database-accessing program, and a single process cache. The facility uses each program cache to store database objects recently retrieved by the cache's database-accessing program, and uses the process cache to store database objects recently retrieved by any of the database-accessing programs. The program caches each allow a database-accessing program to quickly obtain a database object retrieved earlier by the same database-accessing program. The process cache, on the other hand, allows a database-accessing program to quickly obtain a database object recently retrieved by other database-accessing programs. The hierarchy of caches may optionally include additional caches.

Brief Description of the Drawings

Figure 1 is an overview block diagram showing the network of computer systems that the facility preferably operates on.

Figure 2 is a memory diagram showing selective contents of the memories of a database-accessing computer system and the database server computer system.

Figure 3 is a flow diagram showing the steps preferably performed by the facility when a new database-accessing program begins executing in a particular process.

Figure 4 is a flow diagram showing the steps preferably performed by the facility when a database-accessing program is being terminated.

Figure 5 is a flow diagram showing the steps preferably performed by the facility in order to obtain a database object using its object identifier.

Figure 6 is a memory diagram showing the results of performing step 507.

Figure 7 is a memory diagram showing the results of performing steps 510 and 507.

Figure 8 is a flow diagram showing the steps preferably performed by the facility in order to update an object and preference by a specified object identifier.

Figure 9 is a memory diagram showing the results of performing step 802.

Figure 10 is a flow diagram showing the steps preferably performed by the facility in order to update a manipulated object in the database.

Figure 11 is a memory diagram showing the results of performing steps 1001-1005.

Figure 12 is a flow diagram showing the steps preferably performed by the database server in order to update the version of the manipulated object stored in the database to reflect the manipulations to the manipulated object.

Figure 13 is a flow diagram showing the steps preferably performed by the facility on a database-accessing computer system in response to the remote procedure call of step 1202.

Figure 14 is a memory diagram showing the results of performing step 1205.

Figure 15 is a memory diagram showing the results of performing step 1010.

Detailed Description of the Invention

In accordance with the present invention, a database object caching facility ("facility") maintains a hierarchy of caches that enables database-accessing programs processing database transactions to share retrieved database objects across database transactions, reducing process overhead and redundant retrieval from database servers, while still maintaining read-repeatable transactional isolation. (Read-repeatable transactional isolation is a property of database systems according to which a database accessing program can only retrieve committed versions of database objects and no other transaction that modifies any of such objects is permitted to commit until the database-accessing program has completed.) "Database objects" as used herein means any unit of data that may be retrieved from a database including tables, fields, files, programmatic objects, and other units of data. The invention addresses the execution of multiple database-accessing programs, which each produce a stream of related database transactions, each involving one or more database operations.

According to the invention, several database-accessing programs are executed in a single process, allowing database objects and other data to be shared between these programs. In a preferred embodiment, a number of such processes each execute several database-accessing programs. Database-accessing programs each have their own program cache for caching retrieved database objects. The invention preferably maintains the contents of the program cache for subsequent transactions processed by the same program, which have a significant likelihood of using at least some of the objects used by foregoing transactions. Another cache, called the process cache, caches all of the database objects retrieved by any of the database-accessing programs in the process. The process cache, too, is maintained across individual transactions. When a transaction being processed by one of the database-accessing programs needs a database object, the facility first searches the program cache for the database-accessing program and, if the program cache contains the database objects, the database-accessing program accesses it in its program cache. If the database-accessing program's program cache does not contain the needed database object, the database-accessing facility searches the process cache for the needed object, and, if the process cache contains the needed database object, copies it to the program cache where the database-accessing program can access it and make changes to it without affecting the version of the database object used by other database-accessing programs in the same process. If the needed database object is in neither the program nor the process cache, the facility retrieves the needed object from a database server and stores the retrieved database object in both the process cache and the program cache. In a preferred embodiment, as part of reading the database object from the database and copying to the process cache, additional database objects not requested are also read and copied to the process cache, allowing these additional objects to be quickly obtained from the process cache if they are needed for a subsequent transaction.

The facility purges any copies of a database object from all of these caches if notification is received from the database server that the object is being modified by another program, ensuring that any copies of database objects contained by a cache are identical to the database object stored in the database, which in turn ensures read-repeatable transaction isolation.

Access to the program cache and the process cache are each protected by a synchronization mechanism called a lock. As the program cache is accessed almost exclusively by its program, however, its program can obtain this lock quickly in most cases. Therefore, if the program cache is sufficiently large, the time cost of retrieving an object recently used by the same program from the program cache is very small. While the time cost of retrieving an object recently retrieved by a different program in the same process from the process

cache is larger because all of the programs of the process can contend for the lock protecting access to the process cache, this time cost is still significantly smaller than the time cost of using a database server (which often executes on a different machine) to retrieve the object from the database.

Figure 1 is an overview block diagram showing the network of computer systems that the facility preferably operates on. It can be seen from Figure 1 that a network 199 connects a database server computer system 100 with a number of database-accessing computer systems, including database-accessing computer systems 130 and 160. The database computer system 100 maintains the contents of the database, and services requests from all of the database-accessing computer systems to retrieve an update object stored in the database. In a preferred embodiment, additional database server computer systems are connected to the network 199 (not shown) having replicated copies of the database to share the load on the database server computer system 100.

The database server computer system 100 is connected to the network 199 by a network connection 111. The database server computer system further contains an array of central processing units (CPUs) 101, 102, 103, and 104, having processor caches 106, 107, 108, and 109, respectively. The database 121 is stored by the database server computer system on a storage device 121, such as a redundant array of hard disk drives. The database server computer system 100 also contains main memory (memory) 112, which in turn contains database server programs 113 for accessing the database 122, as well as data 114 utilized by the database server programs 113.

A representative one of the database-accessing computer systems is discussed herein. The database-accessing computer system 130 is connected to the network 199 by a network connection 141. Database-accessing computer system 130 contains an array of processors 131, 132, 133, and 134, having processor caches 136, 137, 138, and 139, respectively. The database-accessing computer system further contains main memory (memory) 142. The memory 142 in turn contains a number of process address spaces, including process address spaces 145 and 146. The process address space 145 fully shown is representative of other process address spaces. It contains programs, including the facility 147 and database-accessing programs, such as database-accessing program A 148 and database-accessing program B 149. The process address space 145 further includes data 150 used by the programs of the process address space. The database-accessing computer system 130 further includes a removable media drive 159, which can be used to install software products, including the facility, which are provided on a computer-readable medium, such as a CD-ROM. While the facility is preferably implemented on the connected group of computer systems configured as described above, those skilled in the art will recognize that it may also be implemented on computer systems having different configurations. For example, some of the computer systems may not have features shown herein, or may have features not shown herein. Further, these features may preferably be embodied in other arrangements of computer systems, including being arranged in a single large computer system. Further, the network 199 may be any type of network, including the Internet, intranets, local area networks, wide area networks, and ad hoc systems of connections.

Figure 2 is a memory diagram showing selective contents of the memories of a database-accessing computer system and the database server computer system. Figure 2 shows a portion of the data 250 in the process address space 245 in the memory of the database-accessing computer system 230. The data 150 includes a single process cache 281 having a lock 282. The facility uses the process cache 281 to cache database objects retrieved by any of the database-accessing programs executing in the process having process address space 245. The data 250 further includes a program cache for each data accessing program executing in this process: a program A cache 283 having a lock 284, and a program B cache 285 having a lock 286. Each cache is shown as a table that maps from object identifiers to data that comprises the object referenced by the object identifier. In each case, the existence of this data in a cache is a result of retrieving this data from the database. The caches shown differ from the actual caches preferably used by the facility in a number of minor respects to better illustrate their use by the facility. First, while object identifiers are shown as four-digit decimal numbers, the facility preferably uses object identifiers that are 32-digit hexadecimal numbers in order to ensure that the object identifiers uniquely identify the database objects to which they refer. Also, while the data mapped to from each object identifier may be thousands of bytes long, in each case only one decimal digit of the data is shown. Further, while the caches are shown as simple tables, they are preferably implemented as hash tables that hash from object identifiers to hash table slots containing pointers to the data comprising each object in a variable-length area of the cache. The caches each have a limited size, and preferably discard data that is old and/or no longer being used when their capacity is reached to make room for new data.

Each lock identifies the one thread currently having authority to read or modify the cache to which it corresponds. Each lock preferably contains the thread identifier of the thread having such authority, or a null thread identifier when no thread has such authority. The threads that may have authority to read and modify the shown caches include the threads executing each of the database-accessing programs executing in the process as well as the special thread used by the facility to flush

modified database objects from the caches.

Figure 2 further shows that the process cache 281 contains the data comprising the object in a compact, sequential "streamed" or "flat" form in which the contents of the object are stored in the database and communicated across the network. It can further be seen that the program caches 283 and 285 contain the data comprising each database object in a different "constructed" form, in which the database objects are usable by the accessing programs. This difference is discussed further below.

Figure 2 also shows the database 222 stored in the storage device 221 of the database server computer system 200. The database 222 is shown as a table that maps object identifiers to the data comprising each database object in its streamed form, in which database objects may be transmitted over the network. The database 222 further contains indications of the read and write locks outstanding for each database object, which correspond to authority to read and write the database object, respectively, by database-accessing programs. For each valid object thereafter, the database 222 contains a row that maps from the object identifier to the database object referred to by the object identifier, thereby defining the site of database objects in the database. While the database 222 is shown as a simple table, because of its large size, it is preferably implemented using well-known advantageous indices. It should further be noted that the database 222 and the caches 281, 283, and 285 are shown with incomplete contents to better illustrate their use by the facility.

It can be seen by comparing program A cache 183 and program B cache 185 that objects having object identifiers "1812," "2217," "8760," and "8991" have recently been retrieved from the database by database-accessing program A, while database objects having object identifiers "4079," "2039," "6213," "7170," and "2213" have recently been retrieved from the database by database-accessing program B. It can further be seen that at least a subset of these objects have been stored in the process cache 281. In order to better describe the facility, its operation is discussed herein in conjunction with the series of examples relating to the data 250 shown in Figure 2.

Figure 3 is a flow diagram showing the steps preferably performed by the facility when a new database-accessing program begins executing in a particular process. These steps are preferably executed in the facility by the thread executing the new database-accessing program. In step 301, the facility creates a program cache for the database-accessing program, such as program A cache 283 (Figure 2). In step 302, the facility creates a lock for the program cache, such as program A cache lock 284 (Figure 2). Both steps 301 and 302 preferably involve allocating a required amount of free memory from the process address space for the process in which the database-accessing program is executing. The facility preferably selects the process in which to execute a new database-accessing program in order to group within the same process database-accessing programs likely to access a common group of database objects. These steps then conclude.

Figure 4 is a flow diagram showing the steps preferably performed by the facility when a database-accessing program is being terminated. These steps are preferably executed in the facility by the thread executing the database-accessing program being terminated. In step 401, the facility deallocates the memory allocated to the program cache for the database-accessing program. In step 402, the facility deallocates the memory allocated to the lock for the database-accessing program. These steps then conclude.

Figure 5 is a flow diagram showing the steps preferably performed by the facility in order to obtain a database object using its object identifier. These steps are preferably executed in the facility by a thread executing one of the database-accessing programs. In step 501, the facility obtains the lock 284 for the program cache 283. Step 501 preferably involves determining whether the lock presently contains a null thread identifier and, if so, writing the thread identifier with the current thread into the lock. If the lock does not currently contain a null thread identifier, the facility preferably waits until the lock contains a null thread identifier. The database-accessing program is executing on one of the processors 131-134 of the database-accessing computer system 130 (Figure 1). The first time the database-accessing program obtains the program cache lock, the processor on which the database-accessing program is executing must retrieve the lock from main memory 142. Retrieving the lock from main memory 142 has a relatively high time cost, taking as much time on some processors as is required to execute about 90 instructions. However, as part of retrieving the lock from main memory, the processor stores the lock in its processor cache. For example, processor 131 would store the lock in its processor cache 136. Because the lock is now stored in the processor cache 136, subsequent attempts to obtain the lock by the processor 131 on behalf of this database-accessing program proceed much more quickly because the lock need only be retrieved from the processor cache 131 rather than main memory 142, and retrieval from the processor cache in some processors takes as little time as that required to execute 3 to 12 instructions.

In step 502, after obtaining the lock on the program cache, the facility determines whether the specified object identifier is in the program cache. If so, the facility continues in step 503, else the facility continues in step 505. In step 503, the specified object identifier and data for the database object referenced by the object identifier are already contained in the program cache. For example, program A cache 283 already contains object identifier "2217." In step 503, the facility releases the lock on the program cache. In step 504, the facility returns a pointer to the object in the program cache.

In step 505, the specified object identifier was not in the program cache. For example, object identifiers "2039" and "4444" are not contained in the program A cache 283 (Figure 2). In step 505, the facility obtains the lock 281 on the process cache 282 (Figure 2). Like the program cache lock, the process cache lock must be retrieved from main memory the first time it is obtained. Thereafter it is stored in the processor cache for the processor where it can be accessed quickly for subsequent attempts to obtain the lock by the same database-accessing program. The lock may, however, be migrated to the processor cache of another processor when a database-accessing program executing on the other processor needs to interact with the process cache.

In step 506, after obtaining the lock on the process cache, the facility determines whether the specified object identifier is in the process cache. If so, the facility continues in step 507, else the facility continues in step 509. In step 507, the specified object identifier is in the process cache. For example, the object identifier "2039" is in the process cache 281. In step 507, the facility uses the streamed form of the object stored in the process cache to construct the object in the program cache for the database-accessing program requesting to obtain the object so that the database-accessing program can interact with the database object and its program cache.

Figure 6 is a memory diagram showing the results of performing step 507. It can be seen by comparing Figure 6 to Figure 2 that the facility has constructed the object referenced by object identifier "2039" in program A cache 683. Returning to Figure 5, in step 508, the facility releases the lock 682 on the process cache 681 (Figure 6). After step 508, the facility continues in step 503.

In step 509, the specified object identifier is in neither the program cache for the current database-accessing program nor the process cache, such as object identifier "4444." In step 509, the facility obtains a read lock on the object having the specified object identifier in the database. The database server preferably only permits the facility to obtain a read lock on the object if there are no outstanding write locks on the object. If the object has an outstanding write lock, the facility preferably waits in step 509 until the outstanding write lock is released. In step 510, after obtaining a read lock on the object in the database, a facility copies the streamed form of the object from the database 622 to the process cache 681 (Figure 6). In step 510, the facility preferably further copies a body of read-ahead data from the database containing data comprising additional database objects whose object identifiers were not specified in the object retrieval request. This read-ahead data is preferably also stored in the process cache 681 (Figure 6), so that future requests for the objects contained in the read-ahead data may be serviced from the process cache instead of incurring the cost of accessing the database server computer system across the network and obtaining a read lock on the database object in the database. While steps 509 and 510 as shown accurately portray the overall process of obtaining the object having the specified object identifier from the database, this process is further optimized in a preferred embodiment to account for the relatively large time cost involved in obtaining the object from the database. Because this process takes a relatively long period of time, the facility preferably releases the locks on the program cache and process cache while the retrieval from the database is taking place. Therefore, between steps 506 and 509, the facility releases the program cache lock and the process cache lock (not shown). In step 510, instead of copying the streamed form of the object directly to the process cache as shown, the facility preferably copies the streamed form of the object to a temporary location within the process address space. Then, between steps 510 and 507, the facility obtains the process cache lock, copies the streamed form of the object from the temporary location to the process cache, and obtains the program cache lock (not shown). This optimization allows other database-accessing programs to use the process cache during the retrieval operation. The optimization further allows the first steps shown in Figure 13, described in detail below, to execute against both the process and program caches during the retrieval operation to remove from these cache objects that are about to be modified.

After step 510, the facility continues in step 507 to construct the requested object in the program cache from the streamed form of the object copied into the process cache. The facility constructs objects when they are moved from the process cache to a program cache instead of when they are moved from the database to the process cache in part because of the copying of unrequested read-ahead data comprising additional objects to the process cache as part of retrieving the object from the database. Because objects are only constructed when moved to a program cache, the facility does not incur the overhead of constructing the additional objects until a database transaction needs the additional objects. Figure 7 is a memory diagram showing the results of performing steps 510 and 507. It can be seen by comparing Figure 7 to Figure 6 that the facility has copied the streamed forms of the objects referenced by object identifiers "4444," "4723," "4811," and "4813" from the database 722 to the process cache 781. It can further be seen that the facility has constructed the object referenced by object identifier "4444" in the program A cache 783 using the streamed form of the object in the process cache 781.

In addition to obtaining database objects, database-accessing programs are preferably also able to modify database objects. Figure 8 is a flow diagram showing the steps preferably performed by the facility in order to update an object and preference by a specified object identifier. These steps are executed in the facility

by a thread executing the requesting database-accessing program. In step 801, the facility uses the specified object identifier to obtain the object as shown in Figure 5. After step 801, a current constructed copy of the database object to be manipulated is stored in the program A cache 783, such as the database object referenced by object identifier "4079." In step 802, the facility manipulates the object in the program A cache 783.

Figure 9 is a memory diagram showing the results of performing step 802. It can be seen by comparing Figure 9 to Figure 7 that the facility has manipulated the object referenced by object identifier "4079" in the program A cache 983 by changing its first digit from a "6" to a "5." Returning to Figure 8, in step 803, the facility updates the version of the object stored in the database, as well as any remaining cached versions of the object, to conform with the manipulations to the object in step 802. Step 803 is discussed in greater detail below in conjunction with Figure 10. In step 804, if the object was successfully updated in the database, then these steps conclude, else the facility continues at step 803 to again attempt to update the object in the database. For example, the database server may not have been able to obtain a write lock for the database object.

Figure 10 is a flow diagram showing the steps preferably performed by the facility in order to update a manipulated object in the database in accordance with step 803. In steps 1001-1005, the facility loops through each cache on a local computer system besides the program cache containing the manipulated object. In step 1002, the facility obtains a lock on the current cache. In step 1003, if the current cache contains a copy of the manipulated object, the facility discards the object from the cache. In step 1004, the facility releases the lock on the current cache. In step 1005, the facility loops back to step 1001 to process the next cache.

Figure 11 is a memory diagram showing the results of performing steps 1001-1005. It can be seen by comparing Figure 11 to Figure 9 that the facility has removed the database object referenced by object identifier "4079" from the process cache 1181, as well as from the program B cache 1185. Note, however, that the manipulated object referenced by object identifier "4079" is still contained in the program A cache 1183.

Returning to Figure 10, after all the caches have been processed, the facility continues in step 1006. In step 1006, the facility creates a streamed version of the constructed object in the process cache that was manipulated in step 802 (Figure 8). In step 1007, the facility causes the database server on the database server computer system to update the modified object in the database. Step 1007 preferably involves performing a remote procedure call from the database-accessing computer system to the database server computer system using known remote procedure call protocols. The remote procedure call preferably passes the streamed version of the manipulated object to the database server in the database server computer syst m.

Figure 12 is a flow diagram showing the steps preferably performed by the database server in order to update the version of the manipulated object stored in the database to reflect the manipulations to th manipulated object. The manipulated object is identified by a specified object identifier. In steps 1201-1203, the database server loops through each computer system holding a read lock on the object referenced by the specified object identifier in the database. In step 1202, the database server causes the facility on the current database-accessing computer system to discard from its caches any occurrences of the objects referenced by the object identifier. Step 1202 preferably involves invoking a remote procedure call to the facility of the current database-accessing computer system.

Figure 13 is a flow diagram showing the steps preferably performed by the facility on a database-accessing computer system in response to the remote procedure call of step 1202. In steps 1301-1305, the facility loops through each cache stored on the computer system. In step 1302, the facility obtains the lock on the current cache. In step 1303, if the object is present in the current cache, the facility discards it from the cache. In step 1304, the facility releases the lock on the current cache. In step 1305, the facility loops back to step 1301 to process the next cache on the machine. After all of the caches have been processed, the facility continues in step 1306. In step 1306, the facility releases any read locks it holds on the object in the database. These steps then conclude.

Returning to Figure 12, in step 1203, the database server loops back to step 1201 to process the next database-accessing computer system. In step 1204, the database server obtains a write lock on the object referenced by the object identifier in the database. Step 1204 preferably involves determining that no read locks are outstanding for the database object, which should be true after the performance of steps 1201-1203. Step 1204 further requires that there be no outstanding write locks on the database object. In step 1205, the database server writes the streamed version of the manipulated object to the database for the specified object identifier.

Figure 14 is a memory diagram showing the results of performing step 1205. It can be seen by comparing Figure 14 to Figure 11 that the facility has modified the object stored in the database for object identifier "4079" by changing the first digit from a "6" to a "5." Returning to Figure 12, in step 1206, the database server releases the write lock on the object in the database. In step 1207, the database server returns an indication of whether the manipulated object was successfully written to the database. Step 1207 preferably involves returning the result to the facility on the database-accessing computer system manipulating the object using known remote procedure call protocols.

Returning to Figure 10, in step 1008, if the attempt to update the object in the database succeeded, then

the facility continues in step 1009, else the facility continues in step 1012. In step 1009, the attempt to update the object in the database succeeded, and the facility obtains a lock 1482 on the process cache 1481 (Figure 14). In step 1010, the facility stores the streamed object produced in step 1006 in the process cache 1481 (Figure 14). Figure 15 is a memory diagram showing the results of performing step 1010. It can be seen by comparing Figure 15 to Figure 14 that the facility has stored the manipulated version of the database object in the process cache 1581. In step 1011, the facility releases the lock 1482 on the process cache 1481. In step 1012, the facility returns the result of attempting to update the object in the database.

While the present invention has been shown and described with reference to preferred embodiments, it will be understood by those skilled in the art that various changes or modifications in form and detail may be made without departing from the scope of the invention. For example, the facility may utilize caches at additional levels of the cache hierarchy. For example, the facility may use a computer system cache on each database-accessing computer system to cache objects retrieved by any of the programs in any of the processes on the computer system. Further, the facility may use program group caches to cache objects retrieved by a subset of the programs in one process. The facility may further use different implementations of caches and databases than shown and discussed herein. Indeed, the facility may be gainfully applied to any type of transaction resource managers having concurrent distributed clients, not just to databases. For example, the facility may be applied to such transactional resource managers as a file system, which processes transactions against file system objects, and a compound document management system, which processes simultaneous transactions to modify sections of a compound document. Additionally, the facility may execute database-accessing programs in any number of processes. Also, the database server programs may execute on a data-accessing computer system instead of a dedicated database server computer system. Further, the facility may cache database objects obtained from more than one database server, either in a single combined cache or in separate, per-server caches.

Claims

1. A computer system for processing multiple streams of database transactions in a single process using a hierarchy of caches, comprising:

    a processor that executes programs generating the multiple streams of database transactions in the process; and

    a memory, of which a portion is devoted exclusively to the process in which the programs generating the multiple streams of database transactions are executed, the portion of memory devoted exclusively to the process containing a hierarchy of caches that cache information retrieved in accordance with the streams of database transactions, the hierarchy of caches including:

        a head cache that caches information retrieved in accordance with any of the streams of database transactions, and

        a plurality of leaf caches each caching information retrieved in accordance with one of the streams of database transactions,

        such that information retrieved in accordance with one of the database transactions of one of the streams is stored both in the root cache and in the leaf cache for that stream, and such that retrieving information in accordance with one of the database transactions of one of the streams includes checking the leaf cache for that stream for the information to be retrieved, and, if the leaf cache for that stream does not contain the information to be retrieved, checking the root cache for the information to be retrieved.

2. The computer system of claim 1 wherein, at a given time, because of the limited capacity of the caches, some information earlier cached in one of the caches is no longer contained in the cache.

3. The computer system of claim 1 wherein the hierarchy of caches includes an intermediate cache for a group of two or more streams, such that information retrieved in accordance with one of the database transactions of one of the streams in the group is also stored in the intermediate cache, and such that retrieving information in accordance with one of the database transactions of one of the streams also includes checking the intermediate cache for the information to be retrieved when the leaf cache for that stream does not contain the information to be retrieved.

4. The computer system of claim 1 wherein the processor also executes one or more additional programs generating one or more additional streams of database transactions in a second process,

    and wherein a portion of the memory is devoted exclusively to the second process,

    and wherein the memory further contains a super-root cache that caches information retrieved in accordance with any of the streams of database transactions produced by programs executing in either process, such that information retrieved in accordance with one of the database transactions

of any of the streams of database transactions produced by programs executing in either process is also stored in the super-root cache, and such that retrieving information in accordance with one of the database transactions of any of the streams of database transactions produced by programs executing in either process also includes checking the super-root cache for the information to be retrieved when the root cache does not contain the information to be retrieved.

5. The computer system of claim 1, further comprising a cache information invalidation subsystem that receives indications that information stored in one or more of the caches is no longer valid and, in response, removes the indicated information from the caches, such that information not removed from a cache by the cache information invalidation subsystem may remain in the cache when one transaction in a series of transactions completes and the next transaction in a series of transactions begins.

6. The computer system of claim 1 wherein the database transactions each specify an object to be retrieved, and wherein information retrieved in accordance with the database transactions contains the specified objects, which are retrieved and stored in the root cache in a streamed form in which the programs cannot interact with the retrieved specified objects,

   the computer system further comprising an object construction subsystem that converts retrieved specified objects in the streamed form into a constructed form in which one or more of the programs can interact with the retrieved objects and stores the retrieved specified objects each in the constructed form in the leaf cache for the stream for which the retrieved object was retrieved.

7. The computer system of claim 1 wherein the database transactions each specity an object to be retrieved, and wherein information retrieved in accordance with the database transactions contains the specified objects, which are retrieved and stored in the root cache in a constructed form in which one or more of the programs can interact with the retrieved objects, and wherein the computer system further comprises an inter-cache object copying subsystem that copies the retrieved specified objects in the constructed form from the root cache to the leaf cache for the stream for which the retrieved object was retrieved to enable the program generating the stream for which the retrieved object was retrieved to interact with the retrieved specified objects.

8. The computer system of claim 1 wherein information retrieved and stored in the root cache in accordance with a first database transaction also contains additional objects not specified by the first database transaction, and wherein the object construction subsystem services a second database transaction specifying one of the additional objects by converting the additional object specified by the second database transaction in the streamed form into the constructed form and storing the additional object specified by the second database in the constructed form in the leaf cache for the stream containing the second database transaction.

9. A computer memory containing a segmented caching data structure for caching database objects provided by a database server in response to requests by a plurality of programs, the segmented caching data structure containing:

   a central cache in which are stored copies of database objects provided by the database server in response to requests by any of the programs; and

   for each program, a program cache in which are stored copies of database objects provided by the database server in response to requests by the program, such that, for a request by one of the programs, a copy of the requested object stored in either the central cache or the program cache for the program may be used, obviating provision of the requested database object by the database server.

10. A method in a computer system for processing multiple streams of database operations in a single process using a hierarchy of caches, the hierarchy of caches comprising a single process cache and a stream cache for each of the streams, the method comprising the steps of for each operation in each stream:

   identifying a database object required by the operation;

   determining whether the identified database object is contained in the stream cache for the stream;

   if the identified database object is contained in the stream cache for the stream, using the identified database object contained in the stream cache for the stream; and

   if the identified database object is not contained in the stream cache for the stream:

      determining whether the identified database object is contained in the process cache,

      if the identified database object is contained in the process cache:

copying the identified database object from the process cache to the stream cache for the stream; and

using the copy of the identified database object in the stream cache for the stream,

if the identified database object is not contained in the process cache:

obtaining the identified database object from the database;

storing the identified database object obtained from the database in the process cache;

copying the identified database object from the process cache to the stream cache for the stream; and

using the copy of the identified database object in the stream cache for the stream.

11. The method of claim 10, further comprising the steps of:

receiving an indication that a specified object may be changed;

removing any copies of the specified object stored in the process cache from the process cache; and

removing any copies of the specified object stored in any of the stream caches from the stream caches.

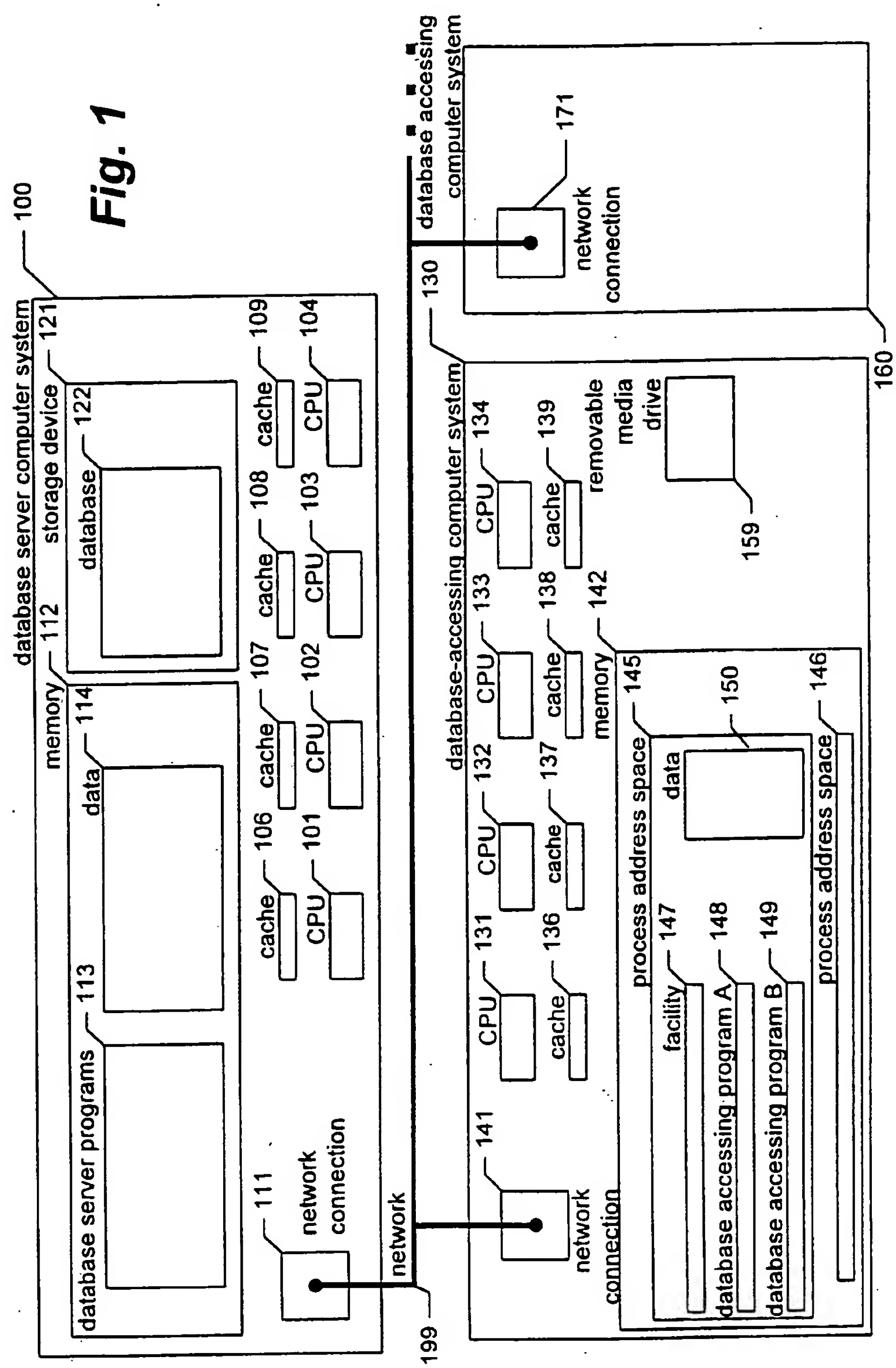
12. The method of claim 10 wherein the obtaining step includes the step of obtaining from the database one or more additional database objects other than the identified database. object, and wherein the storing step includes the step of storing the obtained additional database objects in the process cache.

13. The method of claim 12, further including the steps of:

identifying one of the additional database objects as required by a subsequent operation;

copying the identified additional database object from the process cache for the stream cache for the stream of the subsequent operation; and

using the copy of the additional database object in the stream cache for the stream of the subsequent operation.



100
database server computer system
memory
112
storage device
121
database
122
data
114
database server programs
113
cache
106
107
108
109
CPU
101
102
103
104
network connection
111
network
199
130
database-accessing computer system
CPU
131
132
133
134
cache
136
137
138
139
memory
142
removable media drive
159
process address space
145
data
150
facility
147
database accessing program A
148
database accessing program B
149
process address space
146
network connection
141
database accessing computer system
network connection
171
160

Fig. 1

WEST

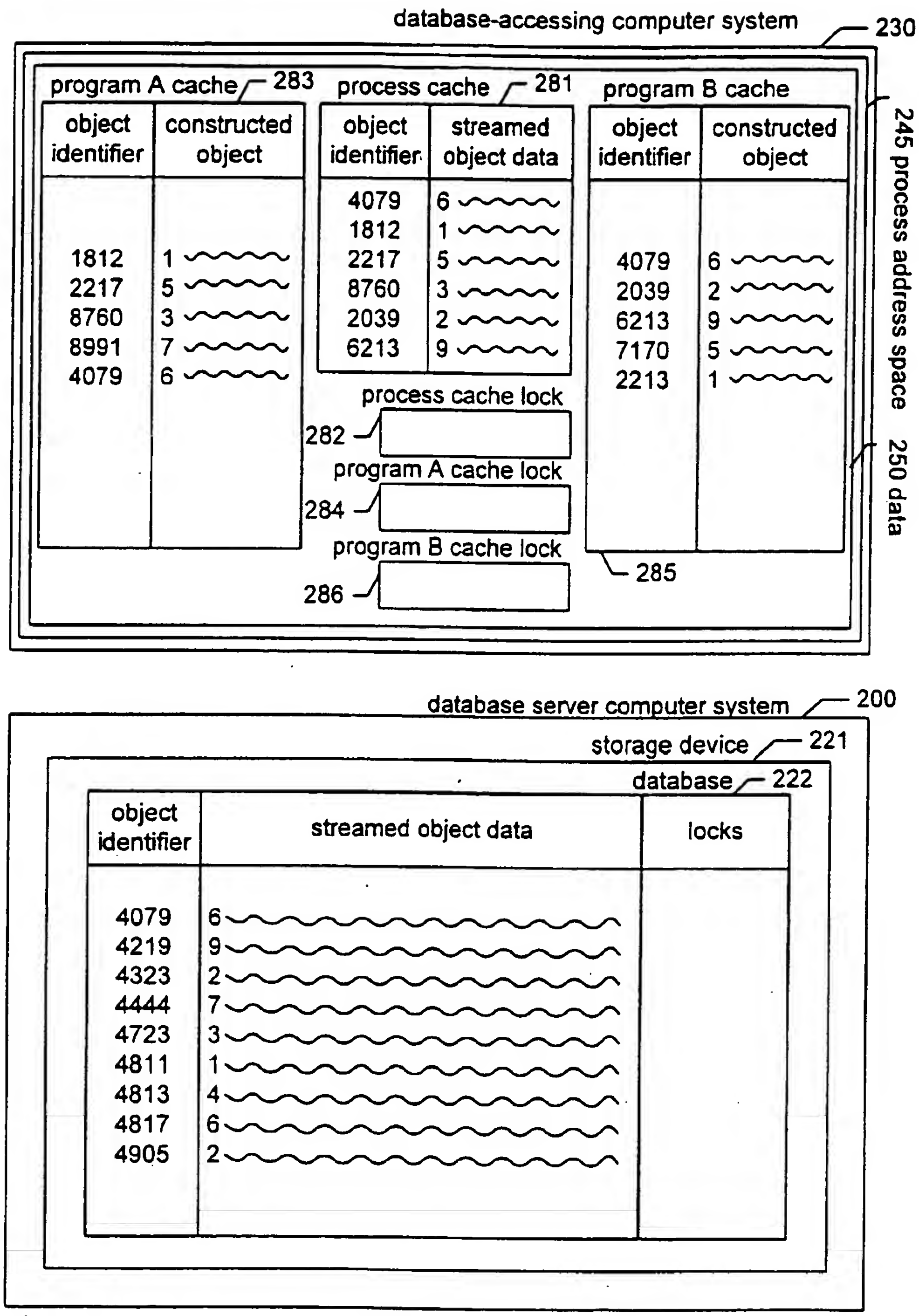

database-accessing computer system
230
program A cache
283
process cache
281
program B cache
object identifier
constructed object
object identifier
streamed object data
object identifier
constructed object
1812 1
2217 5
8760 3
8991 7
4079 6
4079 6
1812 1
2217 5
8760 3
2039 2
6213 9
4079 6
2039 2
6213 9
7170 5
2213 1
245 process address space
process cache lock
282
program A cache lock
284
program B cache lock
286
285
250 data
database server computer system
200
storage device
221
database
222
object identifier
streamed object data
locks
4079 6
4219 9
4323 2
4444 7
4723 3
4811 1
4813 4
4817 6
4905 2

Fig. 2

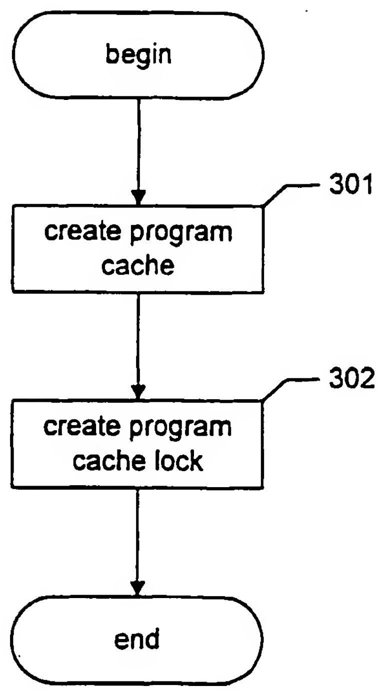

begin
301
create program cache
302
create program cache lock
end

Fig. 3

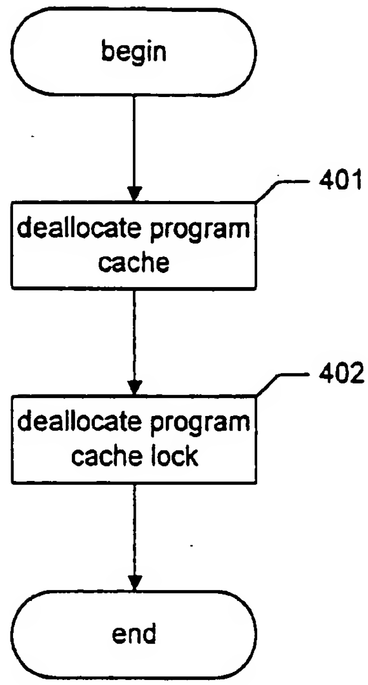

begin
401
deallocate program cache
402
deallocate program cache lock
end

Fig. 4

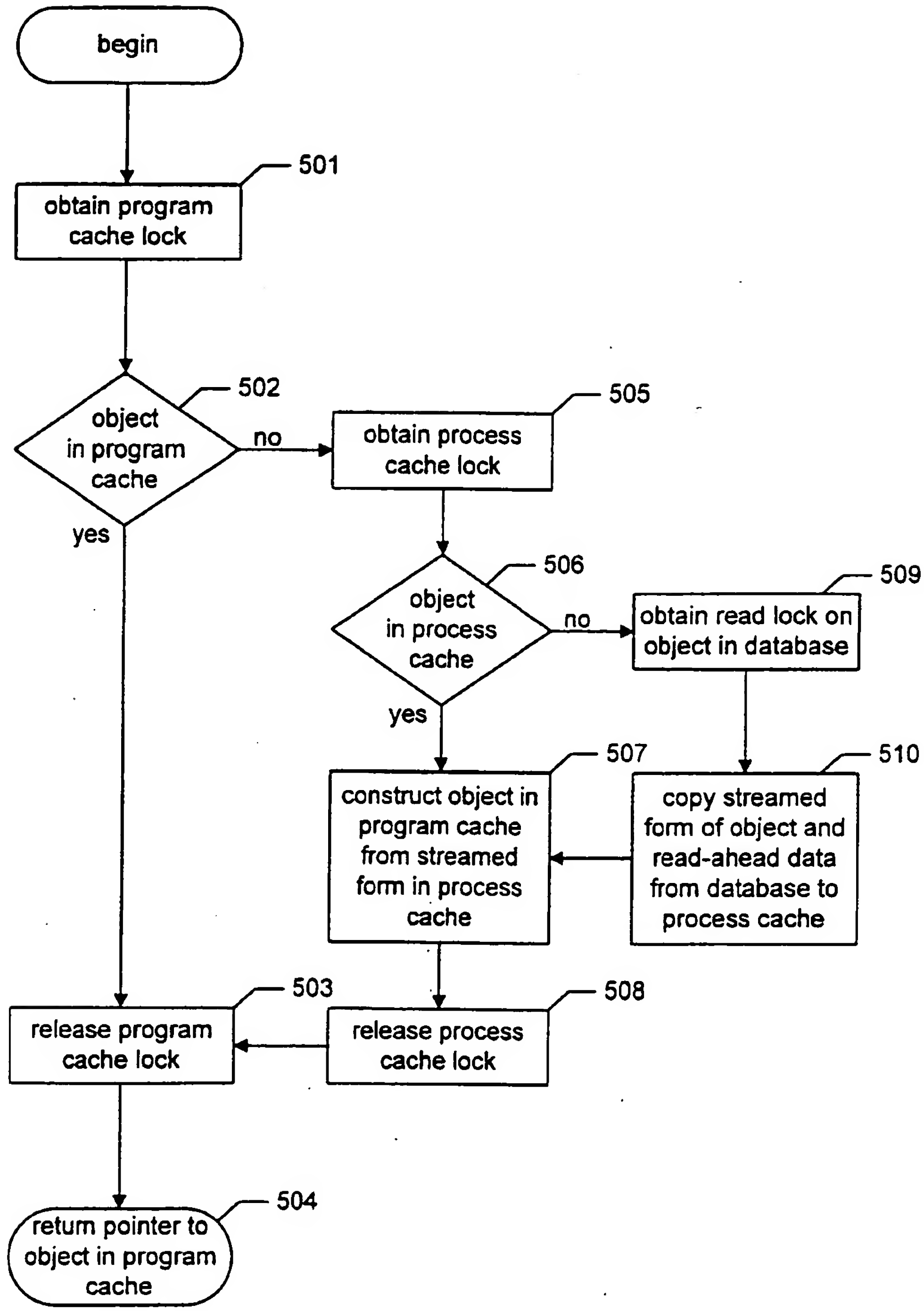

begin
501
obtain program cache lock
502
object in program cache
no
yes
505
obtain process cache lock
506
object in process cache
no
yes
509
obtain read lock on object in database
507
construct object in program cache from streamed form in process cache
510
copy streamed form of object and read-ahead data from database to process cache
503
release program cache lock
508
release process cache lock
504
return pointer to object in program cache

Fig. 5

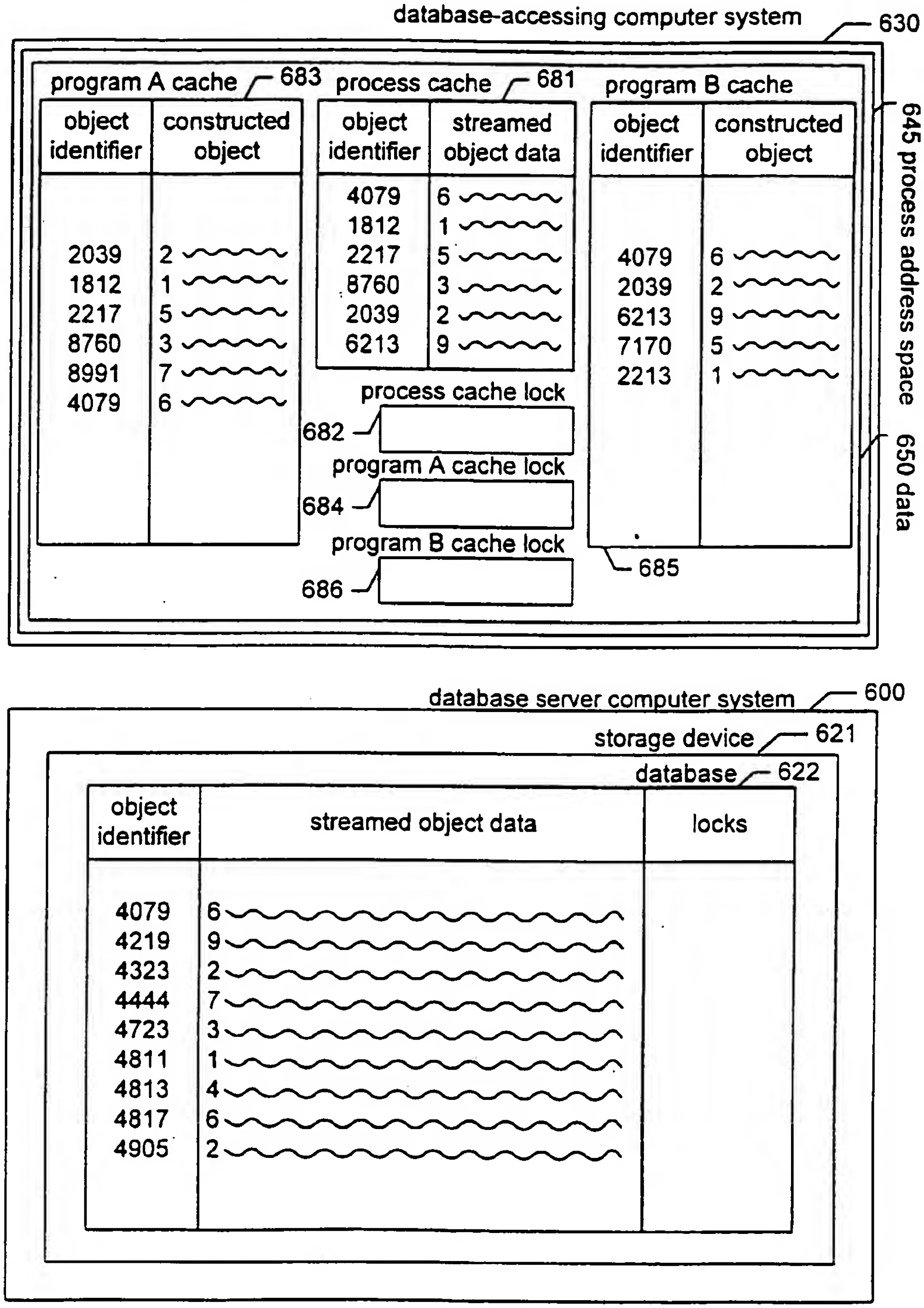

database-accessing computer system
630
program A cache
683
process cache
681
program B cache
645 process address space
650 data
object identifier
constructed object
object identifier
streamed object data
object identifier
constructed object
2039
1812
2217
8760
8991
4079
4079
1812
2217
8760
2039
6213
4079
2039
6213
7170
2213
process cache lock
682
program A cache lock
684
program B cache lock
686
685
database server computer system
600
storage device
621
database
622
object identifier
streamed object data
locks
4079
4219
4323
4444
4723
4811
4813
4817
4905

Fig. 6

WEST

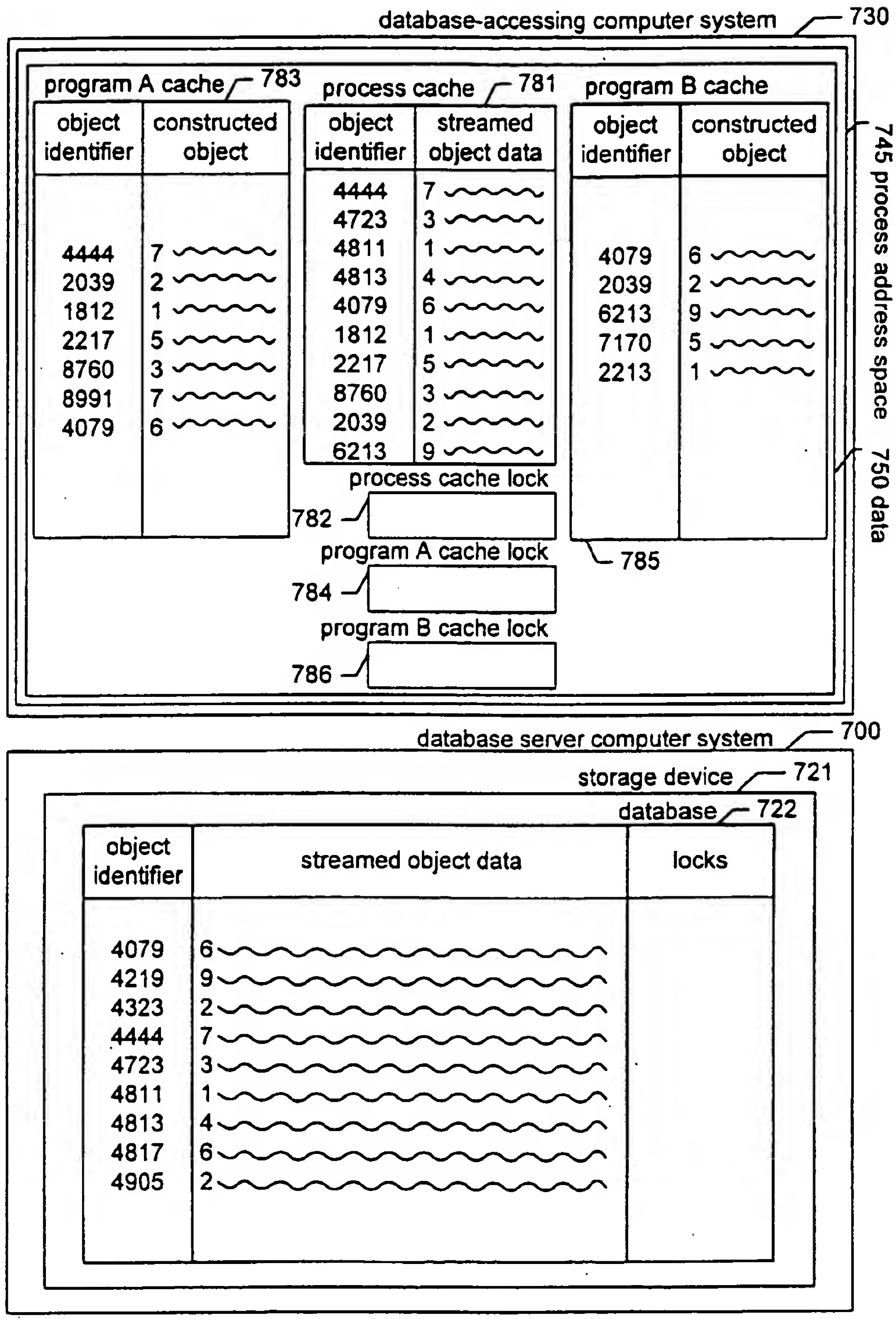

database-accessing computer system
730
program A cache
783
process cache
781
program B cache
745 process address space
750 data
object identifier
constructed object
object identifier
streamed object data
4444
4723
4811
4813
4079
1812
2217
8760
2039
6213
7
3
1
4
6
1
5
3
2
9
4444
2039
1812
2217
8760
8991
4079
7
2
1
5
3
7
6
4079
2039
6213
7170
2213
6
2
9
5
1
process cache lock
782
program A cache lock
784
program B cache lock
786
785
database server computer system
700
storage device
721
database
722
object identifier
streamed object data
locks
4079
4219
4323
4444
4723
4811
4813
4817
4905
6
9
2
7
3
1
4
6
2

Fig. 7

WEST

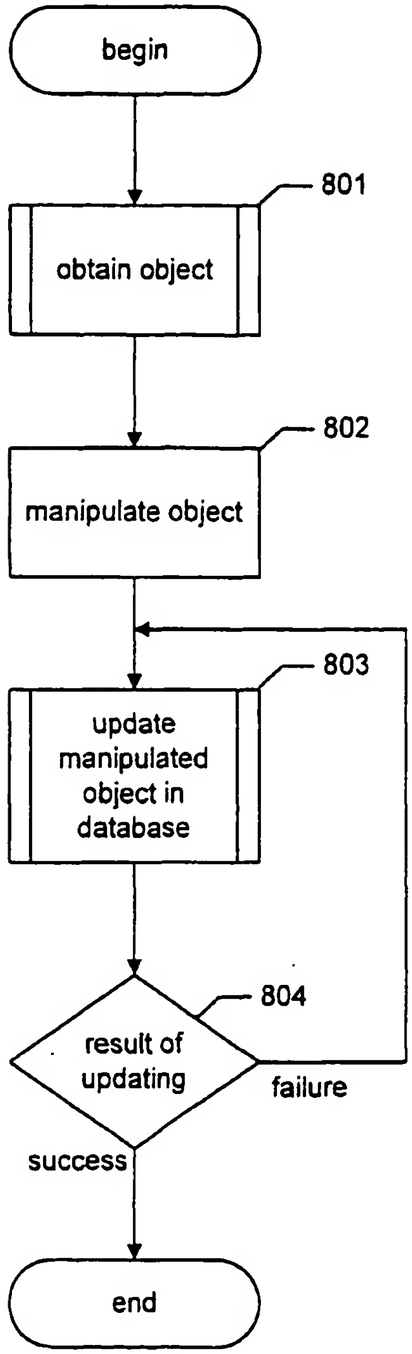

begin
801
obtain object
802
manipulate object
803
update manipulated object in database
804
result of updating
failure
success
end

Fig. 8

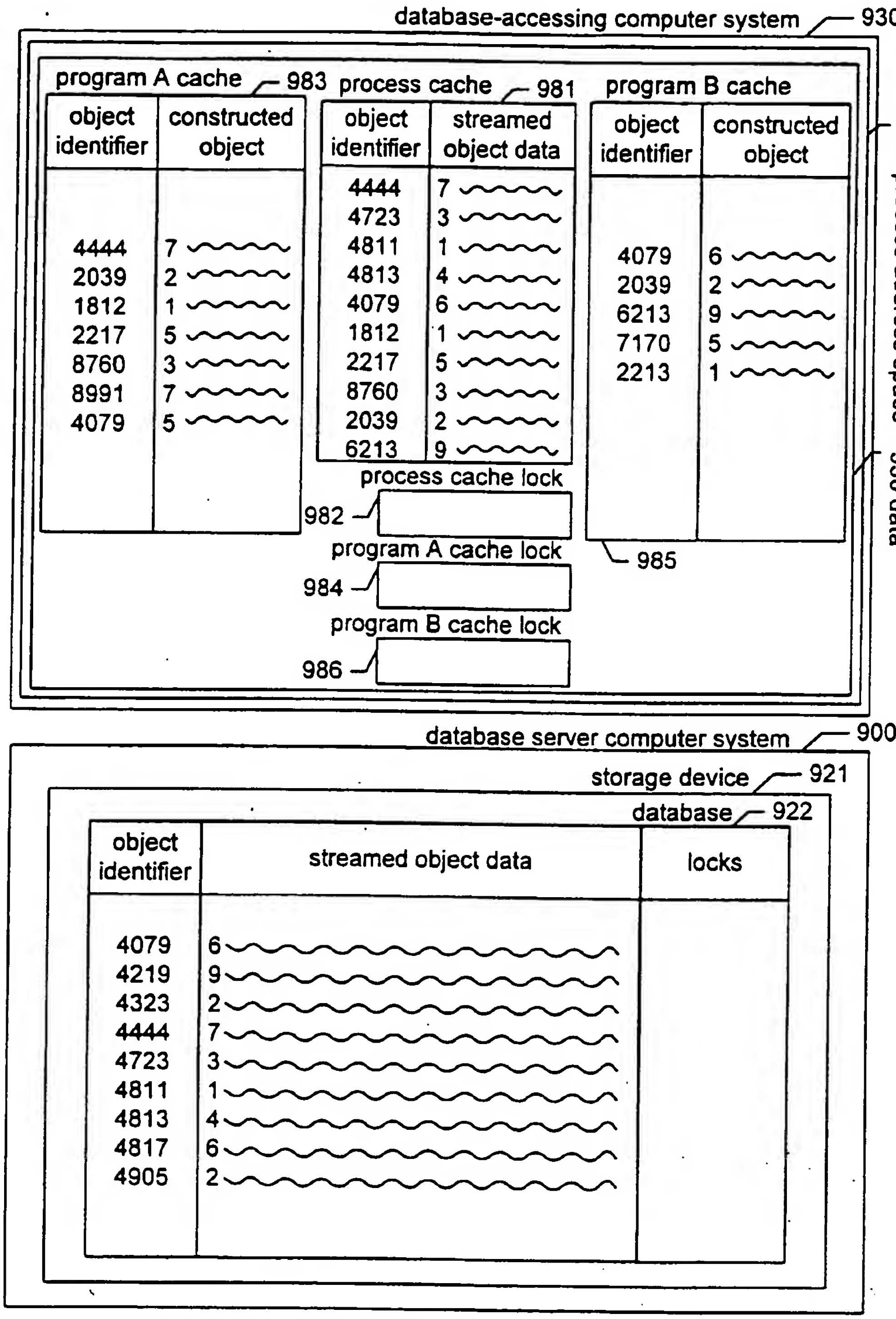

database-accessing computer system
930
program A cache
983
process cache
981
program B cache
object identifier
constructed object
object identifier
streamed object data
object identifier
constructed object
945 process address space
950 data
4444 7
2039 2
1812 1
2217 5
8760 3
8991 7
4079 5
4444 7
4723 3
4811 1
4813 4
4079 6
1812 1
2217 5
8760 3
2039 2
6213 9
4079 6
2039 2
6213 9
7170 5
2213 1
process cache lock
982
program A cache lock
984
program B cache lock
986
985
database server computer system
900
storage device
921
database
922
object identifier
streamed object data
locks
4079 6
4219 9
4323 2
4444 7
4723 3
4811 1
4813 4
4817 6
4905 2

Fig. 9

WEST

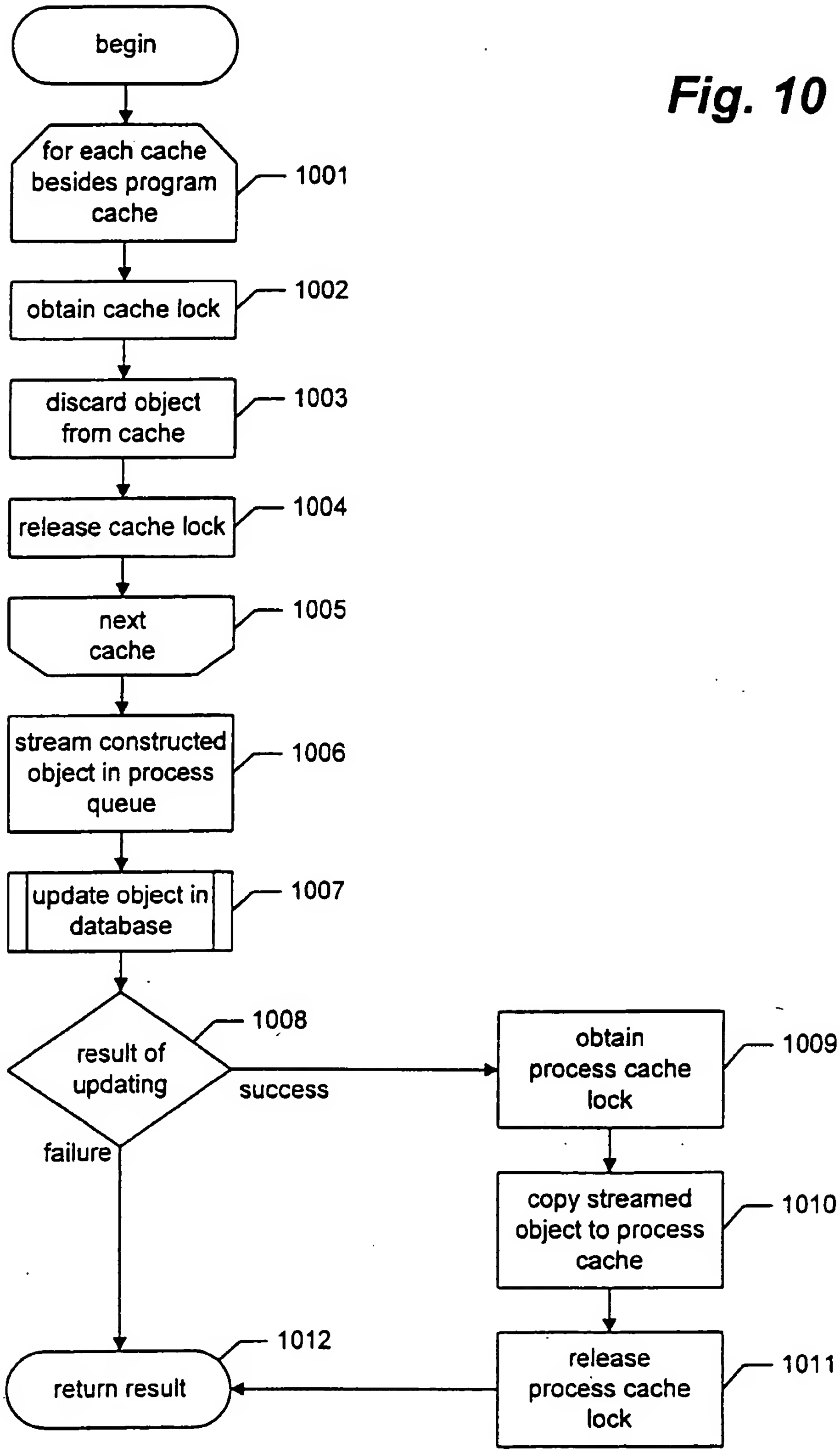

begin
for each cache besides program cache
1001
obtain cache lock
1002
discard object from cache
1003
release cache lock
1004
next cache
1005
stream constructed object in process queue
1006
update object in database
1007
result of updating
1008
success
failure
obtain process cache lock
1009
copy streamed object to process cache
1010
release process cache lock
1011
return result
1012

Fig. 10

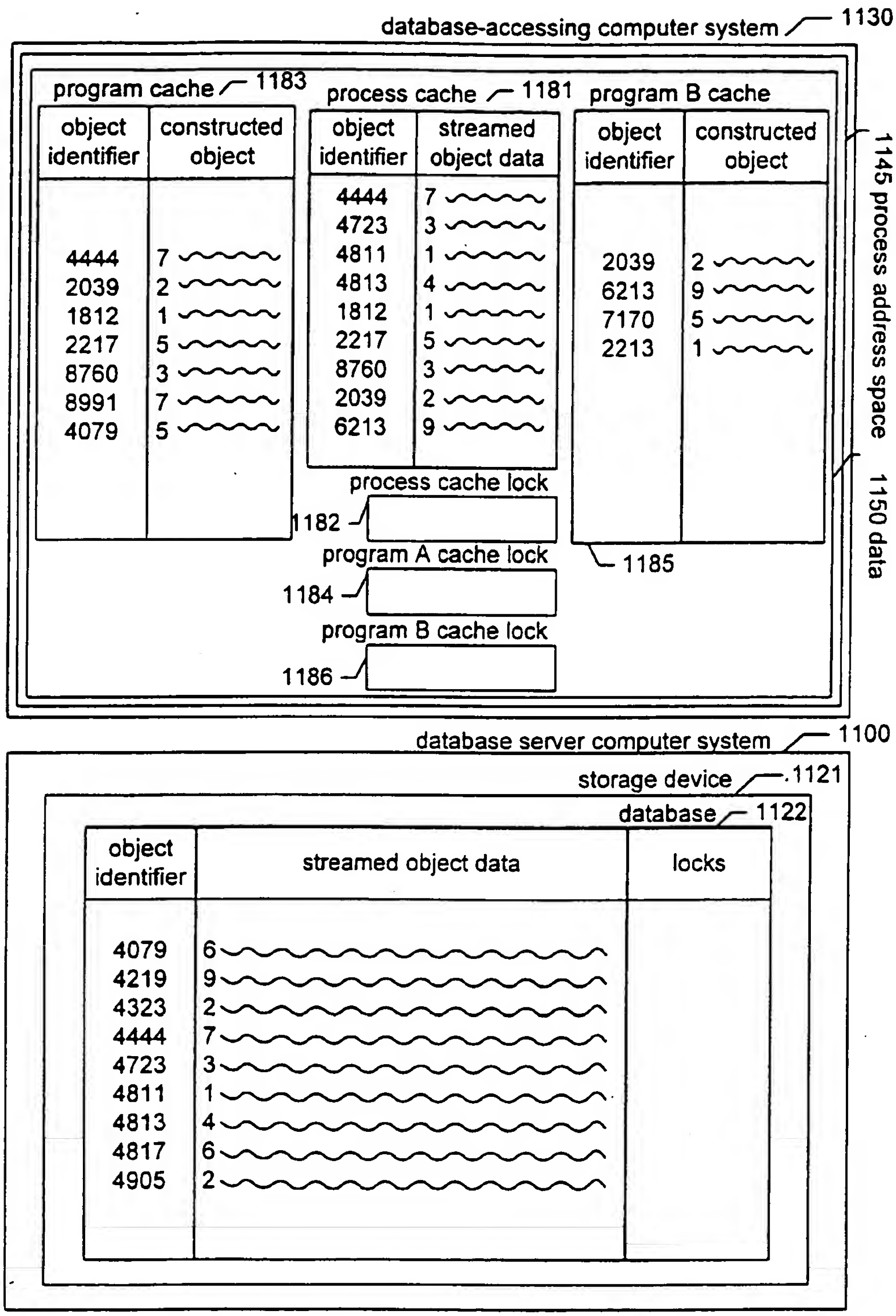

database-accessing computer system
1130
program cache
1183
object identifier
constructed object
4444 7
2039 2
1812 1
2217 5
8760 3
8991 7
4079 5
process cache
1181
object identifier
streamed object data
4444 7
4723 3
4811 1
4813 4
1812 1
2217 5
8760 3
2039 2
6213 9
program B cache
object identifier
constructed object
2039 2
6213 9
7170 5
2213 1
1145 process address space
1150 data
process cache lock
1182
program A cache lock
1184
program B cache lock
1186
1185
database server computer system
1100
storage device
1121
database
1122
object identifier
streamed object data
locks
4079 6
4219 9
4323 2
4444 7
4723 3
4811 1
4813 4
4817 6
4905 2

Fig. 11

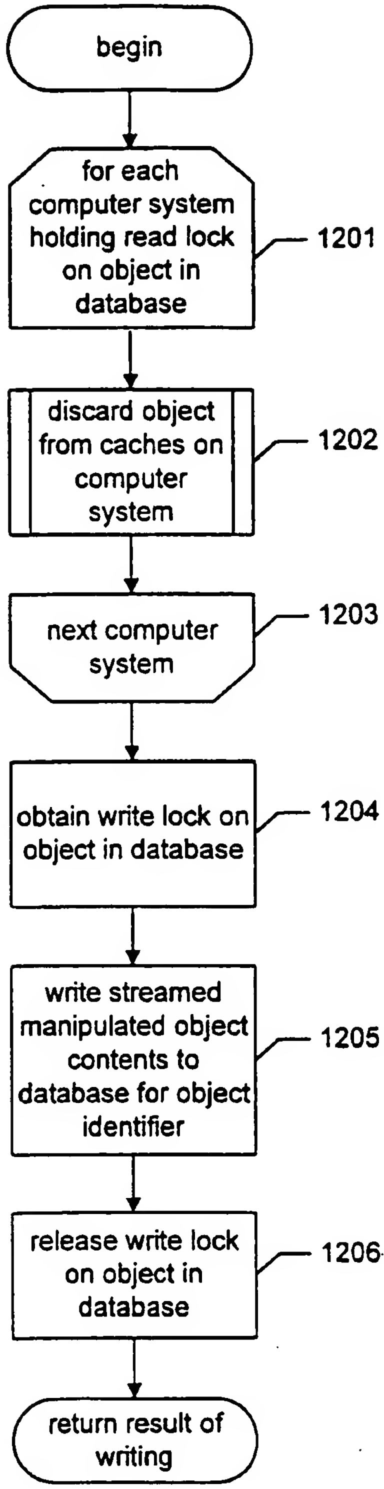

begin
for each computer system holding read lock on object in database
1201
discard object from caches on computer system
1202
next computer system
1203
obtain write lock on object in database
1204
write streamed manipulated object contents to database for object identifier
1205
release write lock on object in database
1206
return result of writing

Fig. 12

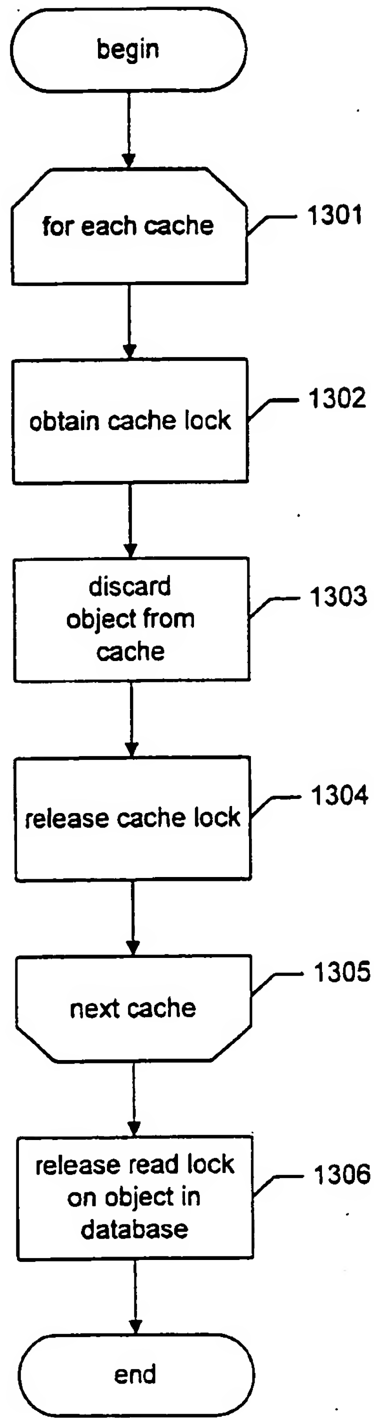
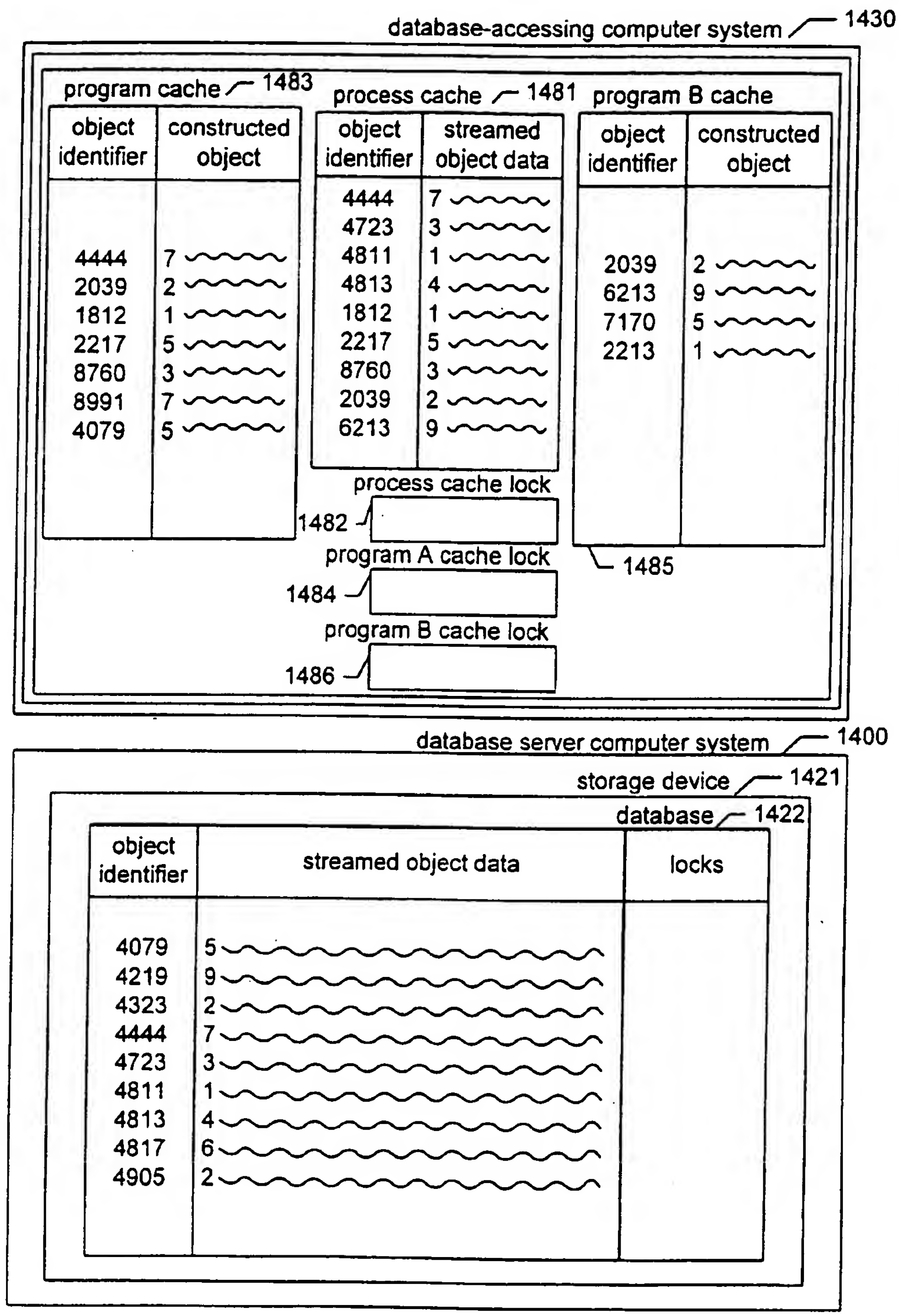

begin
for each cache
1301
obtain cache lock
1302
discard object from cache
1303
release cache lock
1304
next cache
1305
release read lock on object in database
1306
end

Fig. 13



database-accessing computer system
1430
program cache
1483
object identifier
constructed object
4444
2039
1812
2217
8760
8991
4079
7
2
1
5
3
7
5
process cache
1481
object identifier
streamed object data
4444
4723
4811
4813
1812
2217
8760
2039
6213
7
3
1
4
1
5
3
2
9
program B cache
object identifier
constructed object
2039
6213
7170
2213
2
9
5
1
process cache lock
1482
program A cache lock
1484
program B cache lock
1486
1485
database server computer system
1400
storage device
1421
database
1422
object identifier
streamed object data
locks
4079
4219
4323
4444
4723
4811
4813
4817
4905
5
9
2
7
3
1
4
6
2

Fig. 14

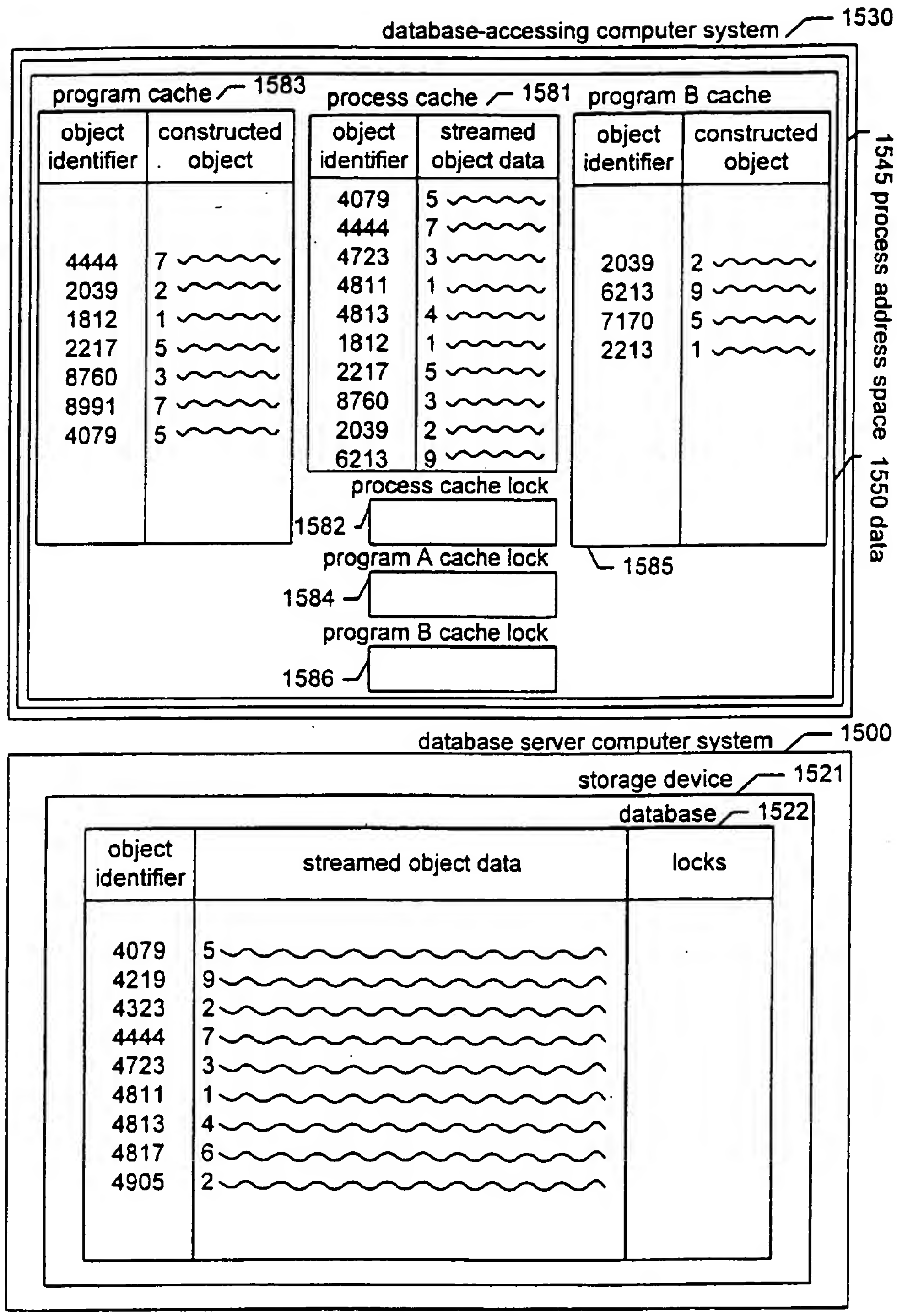

database-accessing computer system
1530
program cache
1583
process cache
1581
program B cache
object identifier
constructed object
object identifier
streamed object data
object identifier
constructed object
4444 7
2039 2
1812 1
2217 5
8760 3
8991 7
4079 5
4079 5
4444 7
4723 3
4811 1
4813 4
1812 1
2217 5
8760 3
2039 2
6213 9
2039 2
6213 9
7170 5
2213 1
1545 process address space
1550 data
process cache lock
1582
program A cache lock
1584
program B cache lock
1586
1585
database server computer system
1500
storage device
1521
database
1522
object identifier
streamed object data
locks
4079 5
4219 9
4323 2
4444 7
4723 3
4811 1
4813 4
4817 6
4905 2

Fig. 15

WEST